Hochschule Merseburg

Fachbereich Informatik und Kommunikationssysteme

Bachelorstudiengang Angewandte Informatik

Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

Entwicklung einer GPS gestützten Augmented Reality Anwendung mithilfe des Wikitude SDK

Eingereicht von: Tobias Franke
Matrikelnummer: 19567

Erstprüfer: Prof. Dr. rer. pol. Uwe Schröter
Zweitprüfer: M.Eng. Ulrich Borchert

Merseburg, 9. Februar 2016

Inhaltsverzeichnis

1 Problemstellung

1.1 Einleitung

Diese Bachelorarbeit befasst sich mit der Entwicklung einer Augmented Reality Anwendung für Android Geräte mithilfe des Wikitude Software Development Kit (SDK) und Global Positioning System (GPS) Tracking.

Der Begriff Augmented Reality, zu deutsch „Erweiterte Realität“, steht für die computergestützte Erweiterung der Realität. Eine der häufigsten Formen von Augmented Reality ist das Überlagern von Bildern oder Videos der realen Welt mit computergenerierten Zusatzinformationen [1].

Mit dem Vorstoß von Smartphones und smarten Gadgets wie Smartwatches und Smartglasses auf den Massenmarkt in den letzten Jahren wurde die technologische Grundlage für Augmented Reality Anwendungen gelegt. Diese sind aus zwei Gründen sehr gut für Augmented Reality geeignet. Einerseits sind die Geräte für den mobilen Einsatz gedacht. Stationäre Anwendungen machen für Augmented Reality nur in wenigen Ausnahmefällen Sinn. Andererseits besitzen sie die notwendigen Sensoren. Diese sind für Augmented Reality Anwendungen zwingend notwendig, denn damit eine Anwendung die Realität erweitern kann, muss sie ihre Umwelt auch erfassen können.

Augmented Reality Anwendungen bestehen aus 3 Hauptkomponenten: Tracking, Visualisierung und Interaktion. Das Tracking ist ein Mittel zur Positionsbestimmung. Die Position wird dabei entweder relativ zu den getrackten Objekten oder absolut mithilfe von GPS, einem Gyrosensor und einem digitalen Kompass bestimmt. Die Visualisierung ist für das Darstellen der Informationen zuständig. Dazu werden die Positionsdaten des Trackings benötigt, um die Informationen am korrekten Platz darstellen zu können. Die Interaktion sorgt dafür, dass der Benutzer mit den dargestellten Objekten interagieren kann.

Der Großteil der Augmented Reality Anwendungen nutzt heutzutage das Objekttracking. Der Grund für die Beliebtheit dieser Methode ist, dass für das Tracking nur eine Kamera notwendig ist. Die Methode hat aber einige entscheidende Nachteile, z. B. ist sie sehr unflexibel, da immer Marker notwendig sind. Auf die Nachteile wird im Abschnitt 2.1.1 noch einmal genauer eingegangen.

In dieser Bachelorarbeit soll eruiert werden, wie weit fortgeschritten das Tracking ohne Kamera ist. Dazu wird die Technik des GPS Tracking verwendet. Durch stetige Weiterentwicklungen der Sensoren könnte das GPS Tracking Augmented Reality Anwendungen einer viel breiteren Masse zugänglich machen. Der große Vorteil von GPS Tracking ist, dass es keine störenden Marker benötigt. Somit können reale und virtuelle Objekte soweit verschmelzen, dass keine Unterschiede mehr wahrnehmbar sind.

1.2 Aufbau der Arbeit

Die Arbeit ist in drei Abschnitte unterteilt: Stand der Technik, Konzeption und Realisierung des Prototypen.

Im ersten Abschnitt soll es um den aktuellen Stand der Technik gehen. Es werden verschiedene Techniken und SDKs verglichen. Entsprechend wird ein SDK und eine Zielplattform ausgewählt.

Der zweite Abschnitt befasst sich mit der Konzeption des Prototypen. Dabei werden die Inhalte der Anwendung definiert, die Architektur und der Programmzyklus erklärt und die App Oberfläche beschrieben.

Im dritten Abschnitt geht es um die Implementierung des Prototypen. Die Implementation wird anhand von Codeauszügen erklärt.

Im Schlusskapitel werden die Erkenntnisse dieser Arbeit zusammengefasst und ein Ausblick auf kommende Technologien gegeben.

2 Stand der Technik

2.1 Aufbau einer Augmented Reality Anwendung

2.1.1 Tracking

Allgemein

In diesem Abschnitt werden die existierenden Techniken des Trackings in Augmented Reality Anwendungen beschrieben. Tracking bedeutet hierbei das Bestimmen der Position eines gesuchten Objektes zu einem Bezugssystem. Die Güte des Trackings wird durch die Abtastfrequenz und durch die Genauigkeit der ermittelten Position bestimmt [14].

Tracking mithilfe von Markern

Das Tracking mit Markern wird verwendet, wenn der Standort des Nutzers keine Rolle spielt. Die Technik nennt man auch bildgestütztes Tracking, da hierbei das von der Kamera aufgenommene Bild analysiert wird.

Die Marker sind in der Software fest definierte Bilder, die mit dem aktuellen Kamerabild abgeglichen und auf Übereinstimmungen überprüft werden. Marker können speziell nur für diesen Zweck gemachte Grafiken sein, die von der Software leicht zu erkennen und zu verarbeiten sind. Ein Beispiel hierfür sind Quick Response (QR) Codes (siehe Abbildung 2.1), eine Grafik bestehend aus weißen und schwarzen Quadraten. Die

Grafik enthält Merkmale, mit denen sich die Rotation und Lage des Markers im Raum eindeutig bestimmen lassen. Die schwarzen und weißen Pixel sorgen durch den hohen Kontrast dafür, dass der QR Code schneller und effizienter von einem Algorithmus erkennbar ist.

Abbildung 2.1: Beispiel für einen QR Code. Inhalt: http://www.hs-merseburg.de/

Es gibt jedoch auch Marker, die nicht speziell für Tracking gemacht wurden. Das können ganz normale Fotos oder Grafiken sein. Diese haben jedoch den Nachteil, dass sie viel aufwändiger und rechenintensiver zu erkennen sind. Dafür bieten sie den Vorteil, dass nicht zwingend Marker für die Anwendung hergestellt werden müssen, sondern auch schon vorhandene Grafiken verwendet werden können. Außerdem können so auf dem Marker selbst auch noch Informationen untergebracht werden.

Bei beiden Arten von Markern gibt es allerdings den Nachteil, dass immer eine Kamera notwendig ist und gute Belichtungsverhältnisse herrschen müssen. Je schlechter die Belichtungsverhältnisse sind, umso geringer werden die Kontraste der Marker und umso schlechter kann der Algorithmus die Marker erkennen.

Tracking ohne Marker

Beim Tracking ohne Marker kann sich die Anwendung nicht an einfach erkennbaren Grafiken orientieren, sondern muss die Umwelt erkennen, analysieren und daraus die

Position des Gerätes ermitteln. Dies erfordert einen bedeutend höheren Rechenaufwand, da man das Bild der Kamera nicht nach speziellen Objekten filtern kann, sondern das gesamte Bild analysieren muss.

Die Erkennung ohne Marker läuft so ab, dass zuerst eine Kantenerkennung in dem Bild durchgeführt werden muss, welche aber sehr aufwändig und teilweise auch fehlerhaft ist. Dann wird versucht zu ermitteln, welche Kanten zusammen gehören. Nachdem man die zu einem Objekt zugehörigen Kanten zusammengefügt hat, erhält man geometrische Objekte, welche man nun mit einer Datenbank abgleichen kann, um die Objekte einordnen zu können.

Diese Herangehensweise hat jedoch auch Schwächen. Beispielsweise funktioniert die Technik nur gut mit Objekten, die eindeutige, klare Konturen haben. Sind die Konturen nicht eindeutig erkennbar, z. B. durch schlechte Lichtverhältnisse, wird die Erkennung schwieriger und somit rechenintensiver bis hin zu unmöglich. Weiterhin funktioniert die Technik nur bei Objekten, die sich in der Datenbank zum Vergleichen befinden. Damit hat man ein ähnliches Problem wie beim Tracking mit Markern, wenn auch nicht so stark ausgeprägt. Damit ein Objekt erkennbar ist, muss es vorher analysiert und in die Datenbank aufgenommen werden. Somit ist auch diese Herangehensweise nicht universell einsetzbar.

Tracking ohne optische Sensoren

Es gibt noch einen weiteren Ansatz für das Tracking in Augmented Reality Anwendungen, der eine komplett andere Vorgehensweise verfolgt. Wenn man das Tracking über optische Sensoren (Kamera) nicht verwenden will oder kann, existiert die Möglichkeit,

die Position und Lage des Gerätes über GPS, einen Gyrosensor[1] und einen digitalen Kompass zu bestimmen.

Dabei bestimmt das GPS die eindeutige Lage im globalen Koordinatensystem und der Gyrosensor bestimmt die Rotation des Gerätes auf der x-, y- und z-Achse. Mit dem digitalen Kompass können die beiden Werte zu einer eindeutigen Position vereint werden. Essentiell hierfür ist das richtige Funktionieren von GPS, Gyrosensor und Kompass. Wenn diese nicht genau sind, kommt es z. B. zu Sprüngen der dargestellten Objekte.

Der große Vorteil der Variante ist die Unabhängigkeit von Markern. Die Position und Rotation können vollkommen autonom ermittelt werden. Das passiert zudem sehr effizient, da keine Rechenleistung für die Bildanalyse verwendet wird. Ebenso funktioniert die Methode auch bei schlechten Belichtungs- oder Sichtverhältnissen.

Der Nachteil der Methode ist die ungenaue Positionierung. Die Genauigkeit der Position ist durch GPS auf bestenfalls 4 m [13] in horizontaler Richtung und 15 m in vertikaler Richtung begrenzt.

2.1.2 Visualisierung

Nachdem mit dem Tracking die Position des Gerätes festgestellt wurde, müssen als nächstes die für Augmented Reality wichtigen Informationen auf dem Display dargestellt werden. Gerade dieser Teil ist charakteristisch für Augmented Reality, denn dabei geschieht der Schritt der „Erweiterung“. Die realen Objekte sollen durch virtuelle Objekte oder Daten erweitert werden. Die Herausforderung hierbei ist, die Objekte

[1] Gyroskopischer Sensor, misst die Rotation auf der x-, y- und z-Achse.

genau an der gewünschten Position anzuzeigen.

Als Grundlage für die Visualisierung dient das Kamerabild. Das wird bei den meisten Augmented Reality Anwendungen in Echtzeit als Hintergrund angezeigt. Es gibt aber auch Anwendungen, die einen eigenen virtuellen Hintergrund generieren. Die virtuell generierten Informationen müssen so über diesem Hintergrundbild angezeigt werden, dass sie zu den Objekten auf dem Kamerabild passen.

Das Visualisieren der digitalen Informationen nennt man auch „Rendern“ (engl.). Rendern bezeichnet das Erzeugen eines Bildes aus Rohdaten.

Bei der Visualisierung kann man zwischen 2D und 3D Objekten differenzieren. Die 2D Visualisierung wird meist verwendet, um zusätzliche Informationen oder Daten über die reale Welt zu legen. Eine 3D Visualisierung ist notwendig, wenn 3D Objekte in die reale Welt eingefügt werden sollen, z. B. bei Spielen.

2D Visualisierung

Die 2D Visualisierung lässt sich relativ einfach realisieren. 2D bedeutet hierbei, dass die Objekte, die die reale Welt erweitern sollen nur zweidimensional sind. Beispiele für solche 2D Objekte sind ein Point of Interest (POI) oder eine Texttafel.

Jedes dieser Objekte hat eine eindeutige Position, durch die es an der richtigen Stelle gerendert werden kann. Die Objekte werden immer so angezeigt, dass diese genau zum Betrachter zeigen. Dadurch ist das Rendern verhältnismäßig ressourcenschonend.

3D Visualisierung

Die 3D Visualisierung ist im Gegensatz zur 2D Visualisierung deutlich rechenintensiver. Da die Visualisierung in Echtzeit passieren muss, kann es gerade hierbei auf leistungsschwacher Hardware zu Performanceeinbrüchen kommen. Das hat zur Folge, dass das Rendern stockt oder ruckelt.

Bei der 3D Visualisierung wird als erstes eine 3D Welt generiert und in den Speicher geladen. Alle Objekte, die gerendert werden sollen, müssen in diese Welt hineingeladen werden. Um ein Objekt in dieser Welt realistisch darstellen zu können, benötigt es 4 Eigenschaften:

- Position
- Rotation
- Skalierung
- Dimension

Jede der Eigenschaften hat jeweils einen x-Wert, einen y-Wert und einen z-Wert, wobei jeder dieser Werte für eine der 3 Dimensionen steht.

Auch der Betrachter wird anhand der beim Tracking des Gerätes ermittelten Position in diese Welt geladen. Er wird dort als eine Art Kamera definiert, die die Objekte aus der berechneten Perspektive betrachtet. Die Kamera hat jedoch nur eine Position und Rotation, aber keine Skalierung oder Dimension. Die Rotation bestimmt hierbei die Ausrichtung, wohin die Kamera schaut.

Zusätzlich kann z. B. noch eine Lichtquelle definiert werden. Die Lichtquelle strahlt Licht aus und beleuchtet die Oberfläche der 3D Objekte. Anhand der Position der Licht-

quelle können die Stärke der Beleuchtung und die Schatten berechnet werden. Dadurch ist eine realistischere Darstellung möglich.

2.1.3 Interaktion

Als Interaktion in einer Augmented Reality Anwendung bezeichnet man die Art und Weise, wie der Benutzer die Anwendung bedient.

Eine Möglichkeit mit der Anwendung zu interagieren ist der Touchscreen. Er kann Tipp- und Wischgesten auf dem Bildschirm erkennen. Zur Verarbeitung der Eingaben wird unter Android die im SDK enthaltene Klasse `GestureDetector` verwendet. Der `GestureDetector` erkennt, wie der Name schon sagt, Gesten. Dazu gehören das einfache und doppelte Tippen auf den Touchscreen, Scrollen und andere Wischgesten. Damit der `GestureDetector` die verschiedenen Eingaben erkennen kann, muss er mit einem Listener[2] initialisiert werden, welcher auf eine Eingabe wartet und die aufgenommenen Eingabedaten an den Handler übergibt. Der Handler analysiert die Eingabe und entscheidet, um welche Geste es sich gehandelt hat. Anhand der erkannten Geste ruft der Handler dann die dafür festgelegte Aktion auf.

Angenommen der Benutzer möchte vertikal scrollen. Er setzt seinen Finger auf den Touchscreen, was der Listener als Startpunkt erkennt und die Pixelkoordinaten des Bildschirms abspeichert. Beim Bewegen des Fingers kann durch Berechnen des Quotienten von Weg- und Zeitdifferenz die Geschwindigkeit der Bewegung bestimmt werden, welche auf die Scrollgeschwindigkeit übertragen wird. Nach dem Loslassen des Touchscreens ist ersichtlich, dass es sich um eine vertikale Scrollbewegung gehandelt hat, da die Differenz der y-Werte des Start- und des Endpunkts größer ist, als die Differenz der x-Werte.

[2]Ein Listener ist eine Funktion, die ein bestimmtes Objekt auf Veränderungen überwacht und bei Eintreten einer Veränderung den zugehörigen Handler aufruft.

Es gibt jedoch verschiedene Listener. Welche Listener verfügbar sind, hängt vom jeweiligen Element der Benutzeroberfläche ab, da jedes Element andere Eigenschaften hat. So kann es sein, dass ein Listener für ein Objekt einfach ungeeignet oder nicht sinnvoll ist. Für Buttons benötigt man z. B. nur einen `OnTouchListener` zum Erkennen, wann der Button gedrückt wird und wann er losgelassen wird, aber keine Listener zum Erkennen von Scroll- oder Wischbewegungen.

Eine weitere, bei Augmented Reality Anwendungen sehr wichtige Möglichkeit zur Interaktion mit der Anwendung, ist das Bewegen des Gerätes. Die Bewegung des Gerätes wird mithilfe von GPS, Gyrosensor und digitalem Kompass bestimmt, wie in Abschnitt 2.1.1 erläutert. Durch das Bewegen kann die in 2.1.2 beschriebene Kamera, welche die 3D Welt aus der Perspektive des Betrachters aufnehmen soll, manipuliert werden.

Das Bewegen des Gerätes im Raum wird dabei vom GPS Modul wahrgenommen und resultiert in der Veränderung der Position der Kamera. Diese kann somit auf der x-, y- und z-Achse bewegt werden. Das Rotieren des Gerätes wird vom Gyrosensor und dem digitalen Kompass erkannt und auf die Rotation der Kamera übertragen. Der Benutzer kann sich also durch Bewegen in der realen Welt auch in der 3D Welt bewegen und durch Umsehen in der realen Welt in der 3D Welt umsehen. Dies ist der entscheidende Punkt, an welchem die reale Welt und die virtuelle Welt verschmelzen und wodurch ein hoher Echtheitsgrad der virtuellen Objekte erreicht wird.

Die Art der Interaktion ist nicht vom Android SDK abgedeckt. Aufgrund dessen ist man bei der Implementation dieser Interaktionsmöglichkeit auf das eingesetzte Augmented Reality SDK angewiesen. Falls auch dies keine Möglichkeit zum Tracking

und zur Visualisierung mittels GPS, Gyrosensor und digitalem Kompass bietet, muss man die Interaktionsmöglichkeit selbst implementieren.

Es gibt noch andere Möglichkeiten der Interaktion, z. B. über die Hardwaretasten des Gerätes, über Bluetooth angeschlossene Eingabegeräte wie Maus und Tastatur oder mittels Hallsensor erfasste Magneten wie bei Google Cardboard [2]. Auf diese Eingabemöglichkeiten wird jedoch nicht weiter eingegangen, da beim Prototypen nur die Interaktion via Touchscreen und Gyrosensor verwendet werden soll (siehe Abschnitt 1.1).

2.2 Vergleich SDKs

Im folgenden Abschnitt soll eine Auswahl an Augmented Reality SDKs in Bezug auf Funktionalität, Umfang und Einschränkungen verglichen werden. Da am Ende des Vergleichs ein SDK für die Entwicklung des Prototypen ausgewählt werden soll, wird bereits eine Vorauswahl getroffen und es werden nur SDKs betrachtet, die die Möglichkeit zum Tracking via GPS bieten.

2.2.1 Wikitude SDK

Das Wikitude SDK ist ein Cross-Plattform[3] SDK, welches Android und iOS unterstützt. Es hat eine sehr gute Application Programming Interface (API) mit vielen Funktionen und bringt Support für 2D und 3D Objekte. Als Interface bietet Wikitude eine Art angepassten WebView. Gestalten kann man diesen mit HTML und CSS. Zugriff auf die API erfolgt auf jeder Plattform einheitlich in JavaScript. Das SDK macht intern aus den JavaScript Funktionen native Funktionen. Es werden Plugins angeboten für Unity, Cordova, Titanium Module und Xamarin. Positiv aufgefallen ist die vollumfängliche API Referenz und die sehr ausführliche Dokumentation mit vielen Beispielen. Weiterhin

[3]Plattformunabhängig, Anwendung kann auf verschiedenen Plattformen ausgeführt werden.

gibt es einen guten Community Support. Zum Testen ist eine kostenlose Vollversion mit Wasserzeichen verfügbar, die Lite und Pro Version kann man zu moderaten Preisen erwerben.

2.2.2 AR Lab

AR Lab ist ebenfalls ein Cross-Plattform SDK, welches Android und iOS unterstützt. Es bietet jedoch nur Unterstützung für 2D Objekte. Die Dokumentation besteht aus einem großen, aber unübersichtlichen Wiki mit viel Beispielcode, jedoch ohne API Referenz. Auch Community Support gibt es nicht. AR Lab ist kostenpflichtig, wobei hier die Kosten pro App anfallen.

2.2.3 AR Media

Auch AR Media ist ein Cross-Plattform SDK für Android und iOS. Es bietet Plug-ins für Unity und OpenSceneGraph an. Support für 2D und 3D Objekte ist vorhanden, jedoch findet man keinerlei Informationen über die Funktionsweise oder die verwendeten Techniken. Das SDK ist schlecht dokumentiert. Es hat keine API Referenz. Eine spärliche Dokumentation ist nur in den Headern der Bibliotheken vorhanden. Ebenso fehlen Erklärungen und Anleitungen zur Verwendung, es gibt nur einige kleine Beispielimplementierungen. Der Community Support ist schlecht. Fragen im Helpdesk werden erst nach mehreren Wochen beantwortet. Die Nutzung von AR Media ist nur über ein teures Abomodell möglich.

2.2.4 Droid AR

Droid AR ist einzig für Android verfügbar. Es ist spezialisiert auf Tracking via GPS. 2D und 3D Support sind gegeben. Der Code ist Open Source und unter GPLv3 lizensiert. Die Implementierung muss in Java erfolgen. Die Dokumentation ist sehr spärlich und

die API ist schwierig zu verwenden.

2.2.5 Metatio SDK

Das Metatio SDK ist ein Cross-Plattform SDK für die Plattformen Android, iOS, Windows und MacOS. Außerdem ist ein Unity Plugin vorhanden. Die Implementierung erfolgt nativ in C++. Es unterstützt Tracking mit und ohne Marker. Das Metatio SDK bietet als Besonderheit LLA Marker. Dies sind Marker, in denen Geokoordinaten gespeichert sind, was der Anwendung ermöglicht, den Standort anhand der Marker zu bestimmen. Trotz der kommerziellen Ausrichtung bietet es eine kostenlose Version mit Wasserzeichen an. Es hat eine umfangreiche API und guten Community Support im Entwicklerportal. Metatio wurde im Mai 2015 von Apple aufgekauft. Seitdem ist es nicht mehr öffentlich verfügbar.

2.3 Auswahl des SDKs

Insgesamt war zu erkennen, dass viele Hersteller versuchen, ihr Produkt besonders gut zu präsentieren und die vorhandenen Funktionen hervorzuheben, damit die Entwickler die meist sehr teuren SDKs kaufen oder ein Abonnement abschließen. Danach ist man jedoch bei den meisten SDKs auf sich allein gestellt, weil die Dokumentationen sehr schlecht bis gar nicht vorhanden sind. Einzig das Wikitude SDK macht es besser. Wegen der sehr guten Dokumentation und der großen Entwickler-Community, die besonders wichtig für die Entwicklung ist, fiel die Wahl auf dieses SDK.

2.4 Wahl des Zielgerätes

Das Wikitude SDK unterstützt eine Vielzahl an unterschiedlichen Plattformen. Darunter befinden sich „Multi-Purpose“ Systeme[4], wie Android und iOS, als auch spezielle, nur für Augmented Reality entwickelte Systeme, wie Google Glass [3], Epson Moverio [4] oder Vuzix M100 [5]. Die Entwicklung des Prototypen soll möglichst ohne zusätzliche Kosten erfolgen, weshalb die dedizierten Augmented Reality Systeme schon einmal ausgeschlossen werden können.

Auch die Wahl zwischen Android und iOS ist nicht schwierig, da für die Entwicklung von iOS Anwendungen zwingend ein Computer mit MacOS benötigt wird, um die Anwendung zu implementieren und zu compilieren. Weiterhin ist für das Programmieren von iOS Apps eine Mitgliedschaft am Apple „Developer Program“ notwendig, welche 99€ pro Jahr kostet. Bei Android hingegen ist die komplette Entwicklung kostenlos. Lediglich für das Anbieten seiner Apps im Play Store ist ein einmalige Registrierungsgebühr von 20$ zu zahlen, was jedoch für dieses Projekt nicht notwendig ist. Auch die Entwicklungssoftware „Android Studio“ ist kostenlos und läuft auf allen gängigen Plattformen (Windows, MacOS, Linux).

Aus den aufgeführten Gründen wurde für die Entwicklung des Prototypen Android als Zielplattform ausgewählt. Android Geräte unterscheiden sich jedoch in ihren Betriebssystemversionen, Bildschirmauflösungen, ihrer Hardware und der Architektur teilweise stark voneinander. Deshalb kann es zur Herausforderung werden, die entwickelte Anwendung auf so vielen Geräten wie möglich problemlos ausführbar zu machen. Es müssen bestimmte Mindestvoraussetzungen definiert werden, die das Zielgerät erfüllt.

Da für die Umsetzung das Wikitude SDK verwendet werden soll, muss das Android

[4]Betriebssystem, das auf verschiedenen Geräten läuft und mehr als einen Zweck erfüllt.

Gerät mindestens den Anforderungen des Wikitude SDKs [6] entsprechen. Darunter zählt, dass auf dem Gerät eine Android Version 4.0 (API Level 14) oder höher läuft. Das Android API Level ist abhängig von der Android Version, die auf dem Gerät installiert ist. Mit jedem Android Versionssprung wurde das API Level um eins erhöht. In diesem Fall wird das minimale API Level auf 15 festgelegt, das heißt die App können nur Geräte mit einer Android Version von 4.0.3 oder neuer ausführen. Wie in der Abbildung 2.2 zu sehen, schließt das ca. 3,8 Prozent aller Android Geräte aus, da diese eine ältere Android Version haben.

Version	Codename	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.9%
4.1.x	Jelly Bean	16	10.0%
4.2.x		17	13.0%
4.3		18	3.9%
4.4	KitKat	19	36.6%
5.0	Lollipop	21	16.3%
5.1		22	13.2%
6.0	Marshmallow	23	0.5%

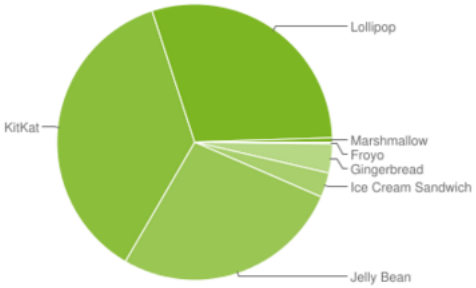

Abbildung 2.2: Verteilung der Android Versionen Stand 07.12.2015 [11]

Weitere Anforderungen des Wikitude SDK sind ein Kompass, ein Beschleunigungssensor und GPS für das Tracking. Um die Umgebung aufnehmen zu können, ist eine rückseitige Kamera erforderlich. Außerdem werden nur „hdpi“-Displays[5] unterstützt. Darüber hinaus muss das Gerät OpenGL 2.0 unterstützen, wodurch sichergestellt wird, dass das Gerät bestimmte Befehle ausführen kann, die zum performanten Ausführen der Anwendung notwendig sind.

[5]High dots per inch, entspricht einer Auflösung von mindestens 240 Pixeln pro Zoll.

3 Konzeption

3.1 Inhalt

In diesem Kapitel soll der allgemeine Entwurf der Augmented Reality Anwendung bestimmt werden. Dabei wird näher auf den Inhalt, den Aufbau und die interne Struktur der Anwendung eingegangen, wie auch die spätere Realisierung konzipiert.

Wie bereits in Kapitel 1 erwähnt, geht es bei dieser Augmented Reality Anwendung hauptsächlich um GPS Tracking. Es sollen die Vorteile einer GPS gestützten Augmented Reality Anwendung aufgezeigt werden, indem verschiedene Möglichkeiten der Darstellung und Interaktion demonstriert werden.

Konkret bedeutet das, folgende Methoden der Visualisierung zu benutzen:

- Positionsmarker für einen POI,
- 2D Objekt in Form eines Labels für einen POI,
- 3D Objekt in Form eines virtuellen Modells.

Mit diesen Objekten soll dann interagiert werden können. Dazu sind drei Aktionen angedacht:

- auf das Objekt tippen,
- auf dem Objekt wischen,
- sich auf/in das Objekt begeben.

Die Visualisierungsmethoden werden mit den Interaktionsmöglichkeiten kombiniert, sodass die folgenden speziellen Anwendungsfälle geplant sind:

- ein POI, der auf Tippen eine Webseite öffnet,
- ein POI mit einer scrollbaren Textbox,
- ein 3D Objekt, das einen Text anzeigt, wenn man sich hindurch bewegt.

3.2 Architektur

Die Realisierung erfolgt mit dem Wikitude SDK, welches ein Cross-Plattform SDK ist. Es besitzt einen JavaScript Kern und plattformspezifische Bibliotheken, welche die JavaScript Funktionen auf native Funktionen abbilden. Diese besondere Architektur soll in dem Abschnitt erläutert werden.

Die Schwierigkeit bei einem Cross-Plattform SDK liegt darin, dass jede mobile Plattform eine andere native Programmiersprache hat. Android Apps werden in Java geschrieben und iOS Apps werden in C# bzw. seit 2014 in Swift [7] geschrieben. Infolgedessen wird eine Sprache gebraucht, die auf allen Systemen unterstützt wird. Das Wikitude SDK nutzt hierzu JavaScript, da es für das Web gebraucht wird und somit auf jeder Plattform verfügbar ist. Neben dem Wikitude SDK nutzt z. B. auch

die mobile Cross-Plattform Entwicklungsumgebung Apache Cordova [8] die Technologie.

Das Wikitude SDK bietet auf jeder Plattform die gleiche JavaScript API, über die die Augmented Reality Funktionen verwendet werden können. Somit kann die gleiche Codebasis für alle Plattformen genutzt werden, was dem Entwickler sehr viel Arbeit spart. Für die Augmented Reality Funktionen ist natürlich auch der Zugriff auf Sensoren und andere Hardware notwendig. Um die Lücke zwischen der Hardware API und der JavaScript Implementierung zu schließen, beinhaltet das Wikitude SDK für jede Plattform zusätzliche Bibliotheken, welche „Computer Vision Engine“ genannt werden (siehe Abbildung 3.1). Die Bibliotheken sind in nativem Code geschrieben, z. B. in Java für Android oder in Swift für iOS, und stellen den JavaScript Funktionen den notwendigen Zugriff auf die Hardware bereit.

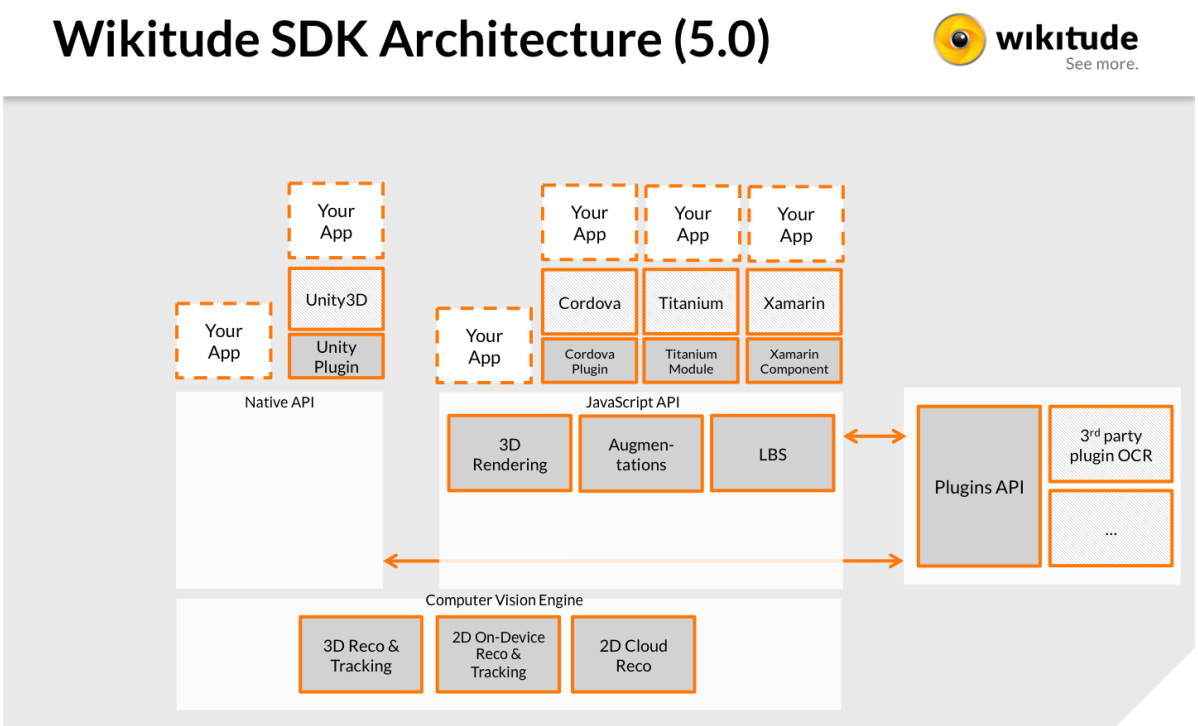

Abbildung 3.1: Architektur des Wikitude SDK [9]

Es ist auch möglich, anstatt der JavaScript API die native Java API zu benutzen. Darauf wird aber nicht weiter eingegangen, weil im Prototypen die JavaScript API verwendet werden soll.

3.3 Programmzyklus

Die Grundstruktur des Programms ist eine Endlosschleife, in der die verschiedenen Berechnungen durchgeführt werden, das Kamerabild mit den gerenderten 3D Objekten überlagert und auf dem Display dargestellt wird. Der in Abbildung 3.2 veranschaulichte Zyklus wird ständig durchlaufen. Hierbei ist es wichtig, dass der Zyklus schnell und oft durchlaufen wird, um einen flüssigen Eindruck zu vermitteln.

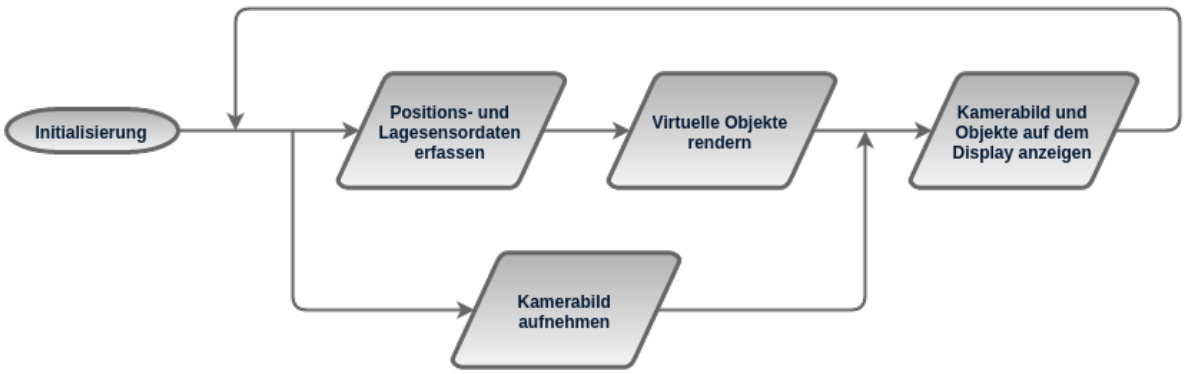

Abbildung 3.2: Grundstruktur des Hauptprogramms

Ab wann eine Bilderfolge flüssig ist, lässt sich nicht allgemeingültig sagen. Bei Kinofilmen wird beispielsweise eine Framerate von 24 Frames per second (FPS) verwendet, beim Fernsehen in Europa 25 FPS. Das reicht bei Filmen gerade so aus, damit es für das menschliche Auge als flüssig erscheint. Jedoch kann es bei Kameraschwenks mit mittlerer Geschwindigkeit zu Rucklern kommen, was versucht wird zu vermeiden, indem nur sehr langsame oder sehr schnelle Kameraschwenks gemacht werden [10]. Da man bei einer Echtzeit Augmented Reality Anwendung die Schnelligkeit der Schwenks nicht

steuern kann, wird festgelegt, dass die Anwendung mit mindestens 30 FPS laufen soll, um Ruckler ausschließen zu können. Das heißt, der Zyklus muss mindestens 30 mal in der Sekunde durchlaufen werden.

Wie in Abbildung 3.2 zu sehen ist, wird das Programm als Erstes initialisiert. Dabei werden die für Augmented Reality benötigten Bibliotheken des Wikitude SDK geladen. Weiterhin wird das Interface der Anwendung geladen, also die Anzeige- und Bedienelemente. Ist das abgeschlossen, geht die Anwendung in die Hauptschleife über.

In der Hauptschleife teilt sich das Programm in zwei Threads[1], wodurch die beiden Threads parallel abgearbeitet werden können. Das parallele Abarbeiten bringt auf einem Mehrkernprozessor im Vergleich zum sequenziellen Abarbeiten erhebliche Geschwindigkeitsvorteile. Der eine Thread holt die aktuellen Daten der Positions- und Lagesensoren und berechnet daraus die Lage des Gerätes. Mit diesen Daten können dann die virtuellen Objekte an ihrer richtigen Position und mit der richtigen Rotation gerendert werden. Der zweite Thread nimmt währenddessen ein Bild mit der Kamera auf. Sind die Objekte gerendert und das Bild aufgenommen, kann wieder in einem Thread weiter gearbeitet werden. Nun geht es darum, die Bilder auf dem Display darzustellen. Dazu dient das Kamerabild als Hintergrund und die virtuellen Objekte werden als Overlay über das Kamerabild gelegt. Das fertige Bild wird dann auf dem Display angezeigt und die Hauptschleife wird wieder von vorn durchlaufen.

[1] Ein Teil eines Prozesses, der auf einem anderen Prozessorkern ausgeführt werden kann.

3.4 App Oberfläche

Unter Android nutzt man für die Umsetzung visueller Elemente sogenannte „Activities“. Eine Activity ist eine Komponente mit einer eigenständigen Oberfläche, welche die sichtbaren Bestandteile, die auch für die Interaktion notwendig sind, enthält. Es kann immer nur eine Activity aktiv sein. Beim Wechseln zwischen den Activities wird die aktuelle Activity inaktiv in den Hintergrund übergeben und die nächste wird aktiviert und in den Vordergrund geholt. Beim Starten einer App wird eine Activity aktiviert. Diese nennt man meist die „MainActivity“.

Die MainActivity im Prototypen ist hauptsächlich dazu da, die Anwendung mit all ihren benötigten Komponenten zu initialisieren. Einer der ersten Schritte ist es, die aktuelle Position vom GPS Sensor abzufragen, sodass dieser sofort damit beginnen kann, seine Position zu ermitteln. Das muss so früh gemacht werden, um möglichst schnell einen GPS Fix[2] zu bekommen und somit die Time to first fix (TTFF) möglichst gering zu halten. Weiterhin hat die MainActivity die Aufgabe, die Bibliotheken des Wikitude SDK zu laden. Währenddessen soll die MainActivity einen Ladebildschirm anzeigen.

Nachdem die MainActivity fertig ist, startet sie eine neue Activity, welche einen ArchitectView enthält. Ein ArchitectView ist eine spezielle Version eines WebView[3], welche im Wikitude SDK enthalten ist. Er ist der Kernbestandteil des Wikitude SDK, da er die Schnittstelle zwischen JavaScript und Java darstellt und die Umgebung, in der die Augmented Reality Funktionen bereitgestellt werden. In der Höhe und in der Breite wird der ArchitectView auf die komplette Activity ausgedehnt, sodass er das

[2] Der Zustand, wenn der GPS Sensor genug Signale empfangen hat, um verlässliche Angaben zur Position zu machen.

[3] Ein Browserframe, in welchem Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript (JS) ausgeführt und angezeigt werden können.

Display vollständig ausfüllt.

Im ArchitectView kann jetzt mit JavaScript der Prototyp realisiert werden. Die Oberfläche wird mit HTML und CSS gestaltet. Darauf wird aber erst später in Kapitel 4 eingegangen, da die Oberfläche während der Laufzeit geändert wird und stark von der aktuellen Situation abhängig ist.

4 Realisierung des Prototypen

4.1 Projektstruktur

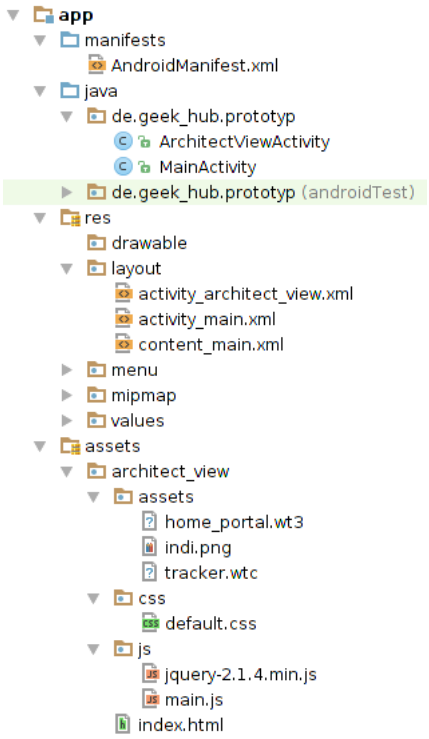

Abbildung 4.1: Projektstruktur mit allen wichtigen Dateien

Das Projekt ist aufgeteilt in zwei Teile. Der erste Teil umfasst das Manifest, den Java Quelltext und die Extensible Markup Language (XML) Layouts. Dieser Teil ist in den Abschnitten 4.2 und 4.3 beschrieben.

Der zweite Teil sind die Assets, also die HTML, CSS und JS Dateien, welche im ArchitectView angezeigt werden. Diese werden in den Abschnitten 4.4 und 4.5 näher erläutert.

4.2 Einbinden des Wikitude SDK

Bevor die eigentliche Implementierung des Prototypen beginnt, muss zunächst ein neues Projekt erstellt werden und die Wikitude Bibliothek eingebunden werden. Das Anlegen eines neuen Android Projektes gestaltet sich mit dem Assistenten des Android Studios sehr einfach und komfortabel. Dabei wird der App Name festgelegt, in diesem Fall schlicht „Prototyp“, das minimale Android API Level auf 15 gesetzt (siehe Abschnitt 2.4) und eine Start Activity ausgewählt.

Nachdem das Projekt angelegt worden ist, wird die Wikitude Bibliothek `wikitudesdk.aar` in das Projekt nach `app/libs/` kopiert. Weiterhin wird im Ordner `app/src/main/` ein Ordner `assets` und gegebenenfalls weitere Unterordner angelegt, in welchen später die Komponenten für den ArchitectView gespeichert werden, also HTML, CSS und JS Dateien sowie Grafiken oder 3D Modelle. Als nächstes wird im Gradle Buildscript[1] `app/build.gradle` die Wikitude Bibliothek eingebunden und der Ordner `libs` als Quelle angegeben:

```
1 dependencies {
2     compile fileTree(dir: 'libs', include: ['*.jar'])
3     testCompile 'junit:junit:4.12'
4     compile (name: 'wikitudesdk', ext:'aar')
```

[1] Tool um den Build Prozess zu managen und zu automatisieren.

```
    compile 'com.android.support:appcompat-v7:23.1.1'
}

repositories {
    flatDir{
        dirs 'libs'
    }
}
```

Listing 4.1: build.gradle: Einbinden der Bibliothek

Außerdem müssen noch im Manifest `app/src/main/AndroidManifest.xml` die benötigten Berechtigungen (`uses-permission`) und Anforderungen (`uses-feature`) definiert werden:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission
    android:name="android.permission.ACCESS_GPS" />
<uses-permission
    android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission
    android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission
```

```
14      android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
15  <uses-feature android:name="android.hardware.camera"
16      android:required="true" />
17  <uses-feature android:name="android.hardware.location"
18      android:required="true" />
19  <uses-feature android:name="android.hardware.sensor.accelerometer"
20      android:required="true" />
21  <uses-feature android:name="android.hardware.sensor.compass"
22      android:required="true" />
23  <uses-feature android:glEsVersion="0x00020000"
24      android:required="true" />
```

Listing 4.2: AndroidManifest.xml: Festlegen der Berechtigungen

Hervorzuheben sind hierbei die Berechtigungen für den Standort und die Kamera, denn diese beiden Sensoren sind die Hauptbestandteile der Anwendung, sowie bei den Anforderungen, dass zwingend eine Kamera, eine Positionsbestimmung, ein Beschleunigungssensor und ein Kompass vorhanden sein müssen. Zusätzlich muss das Gerät OpenGL ES 2.0 (`glEsVersion`) unterstützen. Die benötigten Berechtigungen wurden im Abschnitt 2.4 näher erläutert.

4.3 Anzeigen des ArchitectView

Nachdem das Projekt eingerichtet wurde, folgt als Nächstes das Erstellen und Aufrufen des ArchitectView. Ein ArchitectView ist eine spezielle Art eines WebViews, was bereits näher im Abschnitt 3.4 erläutert wurde. Der ArchitectView kann jedoch nicht direkt angezeigt werden, sondern wird in eine eigene Activity gepackt, die `ArchitectViewActivity`.

Beim Starten einer Android App ist die erste Activity, die geladen wird, immer die `MainActivity`. Sie dient in diesem Fall nur als Ladebildschirm und zeigt den Text „Lädt...“ an, während die `ArchitectViewActivity` geladen wird. Es wird also eine neue Activity mit dem Namen `ArchitectViewActivity` erstellt und als Layout eine leere Activity gewählt. In das XML-Layout in der Datei `activity_architect_view.xml` wird nun der ArchitectView eingefügt:

```
1 <com.wikitude.architect.ArchitectView
2     android:id="@+id/architectView"
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"/>
```

Listing 4.3: activity_architect_view.xml: Definieren des Layout

Durch die `fill_parent` Attribute wird dieser füllend und ohne Ränder dargestellt, einzig die Statusleiste und die Actionbar sind noch am oberen Bildschirm sichtbar.

Um nun die `ArchitectViewActivity` aufzurufen, wird in der `MainActivity` innerhalb der `onCreate()` Methode ein Intent[2] erstellt, diesem die `ArchitectViewActivity` Klasse übergeben und die Activity über den Intent gestartet:

```
1 Intent intent = new Intent(this, ArchitectViewActivity.class);
2 startActivity(intent);
```

Listing 4.4: MainActivity.java: Aufrufen der `ArchitectViewActivity`

[2]Intents werden verwendet, um zwischen Activities zu wechseln und Daten zwischen ihnen auszutauschen.

Nun wird die `ArchitectViewActivity` mit dem darin befindlichen ArchitectView so schnell wie möglich nach dem Start der Anwendung aufgerufen.

Als nächstes folgt das Vorbereiten des ArchitectView, also das Starten der Standortbestimmung, das Managen des Android „Lifecycle“ und letztendlich das Laden des ArchitectView. Dazu werden als erstes zwei Klassen des Wikitude SDK importiert, um den ArchitectView erfolgreich initialisieren zu können:

```
1 import com.wikitude.architect.ArchitectView;
2 import com.wikitude.architect.StartupConfiguration;
```

Listing 4.5: ArchitectViewActivity.java: Importieren der Wikitude Klassen

Die Lifecycle Callbacks für `onResume()`, `onPause()`, `onDestroy()` und `onLowMemory()` haben alle den gleichen Aufbau. Zuerst wird die jeweilige `super`-Methode aufgerufen und dann die entsprechende `architectView`-Methode. Für `onResume()` sieht das zum Beispiel so aus:

```
1 @Override
2 protected void onResume() {
3     super.onResume();
4     if ( this.architectView != null ) {
5         this.architectView.onResume();
6     }
7 }
```

Listing 4.6: ArchitectViewActivity.java: onResume()

Innerhalb des `onCreate()` Callback wird die Standortbestimmung durchgeführt. Da-

zu wird ein `LocationManager` angelegt, welcher mit einem Handle zum Standortservice auf Systemebene initialisiert wird. Außerdem wird ein `locationListener` angelegt, der alle Positionsänderungen an den ArchitectView weiter gibt. Der `locationManager` fordert dann vom GPS Modul (`LocationManager.GPS_PROVIDER`) fortlaufend Updates an und übergibt diese dem `locationListener`:

```
1  LocationManager locationManager = (LocationManager)
2          this.getSystemService(Context.LOCATION_SERVICE);
3
4  this.locationListener = new LocationListener() {
5      public void onLocationChanged(Location location) {
6          if (location!=null) {
7              ArchitectViewActivity.this.lastKnownLocaton = location;
8
9              if ( ArchitectViewActivity.this.architectView != null ) {
10                 if (location.hasAltitude() &&
11                     location.hasAccuracy() &&
12                     location.getAccuracy()<7) {
13
14                   ArchitectViewActivity.this.architectView.setLocation(
15                         location.getLatitude(),
16                         location.getLongitude(),
17                         location.getAltitude(),
18                         location.getAccuracy() );
19                 } else {
20                   ArchitectViewActivity.this.architectView.setLocation(
21                         location.getLatitude(),
```

```
22                          location.getLongitude(),
23                          location.hasAccuracy() ?
24                                  location.getAccuracy() : 1000 );
25                  }
26              }
27          }
28      }
29      public void onStatusChanged(String provider,
30          int status,
31          Bundle extras) {}
32      public void onProviderEnabled(String provider) {}
33      public void onProviderDisabled(String provider) {}
34  };
35  locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
36      0,
37      0,
38      this.locationListener);
```

Listing 4.7: ArchitectViewActivity.java: Initialisieren des `locationListener` und `locationManager`

Zuletzt wird im `onCreate()` Callback noch der ArchitectView mit dem SDK Key[3] konfiguriert:

```
this.architectView =
    (ArchitectView)this.findViewById(R.id.architectView);
final StartupConfiguration config =
    new StartupConfiguration(WIKITUDE_SDK_KEY);
this.architectView.onCreate(config);
```

Listing 4.8: ArchitectViewActivity.java: Konfigurieren des ArchitectView

Im `onPostCreate()` Callback wird schlussendlich der ArchitectView mit der HTML Datei aus den Assets (`app/src/main/assets/architect_view/index.html`) geladen:

```
@Override
protected void onPostCreate( final Bundle savedInstanceState ) {
    super.onPostCreate(savedInstanceState);
    if ( this.architectView != null ) {
        this.architectView.onPostCreate();
        try {
            this.architectView.load("architect_view/index.html");
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }
}
```

Listing 4.9: ArchitectViewActivity.java: onPostCreate()

[3]Notwendig für die Lizensierung.

Damit ist die Java-seitige Implementierung soweit abgeschlossen. Das restliche Programm wird innerhalb des ArchitectView mithilfe von HTML, CSS und JS implementiert. Alle benötigten Daten der Hardwaresensoren werden von der Activity an den ArchitectView geleitet oder vom Wikitude SDK direkt verarbeitet.

4.4 Inhalte des ArchitectView

Nachdem im letzten Abschnitt erläutert wurde, wie der ArchitectView geladen wird, soll es in diesem Abschnitt darum gehen, was im ArchitectView selbst passiert. Der ArchitectView, der zum Teil aus einem WebView[4] besteht, muss nun die Webseite laden, die `index.html`. Die HTML-Datei sowie alle weiteren dafür benötigten Dateien wie CSS-Dateien, JS-Dateien und 3D-Modelle befinden sich im Assets Ordner.

Wie in Listing 4.10 am Dokumentanfang zu sehen ist, wurde für die Webseite der Doctype html gewählt, was dem aktuellen HTML5 Standard entspricht. Die Anwendung setzt somit auf die modernsten Web-Technologien und ist damit bestens für die Zukunft gewappnet. Weiterhin ist zu sehen, dass der Viewport[5] festgelegt wird. Dieser ist initial so, dass die Webseite von der Größe her 1:1 auf dem Display angezeigt wird und die Breite des Viewports genau der Breite des Gerätes entspricht.

```
1 <!DOCTYPE HTML>
2 <html>
3 <head>
4     <meta http-equiv="Content-Type" content="text/html;charset=UTF-8"/>
5     <meta content="width=device-width,initial-scale=1" name="viewport">
```

[4] Ein Browser Frame, welcher direkt in einer App angezeigt werden kann.
[5] Auf dem Bildschirm sichtbarer Bereich der Webseite.

```
6
7       <title>Prototyp</title>
8
9       <script src="js/jquery-2.1.4.min.js"></script>
10      <script src="architect://architect.js"></script>
11      <script src="js/main.js"></script>
12
13      <link rel="stylesheet" href="css/default.css">
14  </head>
15  <body>
16      <div id="loadingMessage" class="info">Loading ...</div>
17      <div id="poiDescription">
18          <div id="content">
19              <h2>Polymerisationskessel</h2>
20              <p>Reduktionsgefäße dieser Art dienen u. a. zur [...]</p>
21              <p>Weltweit werden heute über 20 Mill. t pro Jahr [...]</p>
22              <p>Mit dieser Menge könnte man ein Druckrohr von [...]</p>
23          </div>
24          <div id="button">
25              <input type="button" class="btn" value="Schließen"
26                  onclick="togglePoiDescriptionFullscreen()">
27          </div>
28      </div>
29  </body>
30  </html>
```

Listing 4.10: index.html: Struktur der Webseite

Als nächstes werden die JS Bibliotheken geladen. Dies ist zum einen die Open Source Bibliothek jQuery [12], die komfortable Funktionen zur einfacheren Manipulation von Webseiten und viele weitere Funktionen bereitstellt. Die jQuery Biliothek wird aus dem js Unterordner geladen. Zum anderen wird an dieser Stelle die `architect.js` Datei eingebunden. Sie befindet sich nicht direkt im Assets Ordner, sondern wird über ein spezielles Protokoll direkt vom Wikitude SDK geliefert. Als letzes Script wird die `main.js` Datei eingebunden. Darin befindet sich die Anwendungslogik des Prototypen, welche im Abschnitt 4.5 näher erläutert wird.

Der Body[6] der HTML-Datei besteht nur aus zwei Elementen, einem „Laden“-Hinweis und der Beschreibung für den POI. Die „Laden“-Info wird standardmäßig beim Starten der App angezeigt. Ist alles fertig geladen, wird sie ausgeblendet. Die Beschreibung hingegen ist durch entsprechende CSS Attribute nicht sichtbar.

4.5 Anwendungslogik

4.5.1 Aufbau

Das `main.js` Script enthält die gesamte Anwendungslogik, in der die Eigenschaften der virtuellen Objekte festgelegt werden und die Objekte in den ArchitectView geladen werden. In der ersten Zeile der Datei kann man sehen, dass der komplette Code in einer anonymen Funktion gekapselt ist. Diese Funktion ist der Event Handler für das jQuery `ready()` Event, das heißt die Funktion wird erst ausgeführt, nachdem die Seite komplett geladen und jQuery fertig initialisiert wurde.

[6] Sichtbarer Teil der Webseite.

4.5.2 Deklaration

Am Anfang des main.js Scriptes wird das `poi`-Array deklariert. Es spielt eine wichtige Rolle, denn darin werden alle virtuellen Objekte gespeichert. Die Elemente des Arrays sind Objekte, in denen die Eigenschaften und die Instanzen der virtuellen Objekte gespeichert sind. Die anzuzeigenden, virtuellen Objekte sind auch im Programm als Objekte abgebildet.

4.5.3 Funktionen

Initialisierung

Die erste Funktion heißt `initialisation()`. In ihr wird definiert, welche POIs es geben soll und wie deren Eigenschaften sind. Wie im Abschnitt 3.1 beschrieben, sollen 3 POIs mit unterschiedlichem Verhalten angelegt werden. Als erstes wird der POI initialisiert, welcher eine Webseite öffnen soll. In diesem Fall wird der POI auf die Position der Mensa der Hochschule Merseburg gelegt und mit einem Klick auf ihn soll der aktuelle Speiseplan geöffnet werden.

```
1  poi.push({
2      location: new AR.GeoLocation(51.343362, 11.976180),
3      content: new AR.Label(" Mensa - Speiseplan ",
4          1, {
5              zOrder: 1,
6              onClick: function() {
7                  AR.context.openInBrowser(
8                      'http://meine-mensa.de/speiseplan'
9                  );
10             },
```

```
11              style: {
12                  textColor: '#FFFFFF',
13                  backgroundColor: '#3F51B5',
14                  fontStyle: AR.CONST.FONT_STYLE.BOLD
15              }
16          }),
17      indicator: true
18  });
```

Listing 4.11: main.js: POI zum Anzeigen einer Webseite initialisieren

Beim Initialisieren wird ein Objekt mit den gewünschen Eigenschaften festgelegt und dieses in das `poi`-Array gepusht. Die erste Eigenschaft ist die Position `location` vom Typ `AR.GeoLocation`. Die Klasse `AR.GeoLocation` ist aus dem Wikitude SDK und definiert eine Position durch GPS Koordinaten. An dieser Position wird später das virtuelle Objekt dargestellt.

Die zweite Eigenschaft ist der Inhalt `content`. Mit ihm wird bestimmt, wie das virtuelle Objekt im ArchitectView angezeigt werden soll, in diesem Fall als Label. Das Label ist vom Typ `AR.Label` und enthält verschiedene Präferenzen zum Aussehen des Labels und einen `onClick`-Listener. Diesem `onClick`-Listener wird eine anonyme Funktion übergeben, in der die Methode `AR.context.openInBrowser()` des SDK verwendet wird, um die angegebene Adresse mit dem Webbrowser zu öffnen.

Die letzte Eigenschaft des `poi`-Objektes ist der Indikator. Ein Indikator ist eine visuelle Hilfe für den Benutzer, die anzeigt in welcher Richtung sich ein POI befindet, falls er außerhalb des sichtbaren Bereiches ist. Durch setzen auf `true` oder `false` kann der Indikator aktiviert bzw. deaktiviert werden.

Der zweite POI soll durch Tippen einen scrollbaren Text öffnen. Dafür wurde der Polymerisationskessel ausgewählt, welcher sich neben dem Hauptgebäude der Hochschule Merseburg befindet. Durch Tippen auf den POI wird ein Infotext mit näheren Informationen zu diesem Objekt angezeigt.

```
poi.push({
    location: new AR.GeoLocation(51.343525, 11.977697),
    content: new AR.Label(" Polymerisationskessel ",
        1, {
            zOrder: 1,
            onClick: togglePoiDescriptionFullscreen,
            style: {
                textColor: '#FFFFFF',
                backgroundColor: '#3F51B5',
                fontStyle: AR.CONST.FONT_STYLE.BOLD
            }
        }),
    indicator: true
});
```

Listing 4.12: main.js: POI der einen Infotext öffnet

Wie in Listing 4.12 zu sehen ist, ähnelt die Initialisierung der des ersten POI. Es wird jedoch eine andere Funktion aufgerufen beim Klick auf diesen POI. Die Funktion `togglePoiDescriptionFullscreen` wird später im Abschnitt „Infotext anzeigen“ erläutert.

Der dritte POI soll ein 3D Objekt sein, das beim Hindurchbewegen einen Text anzeigt. Für dieses Beispiel wurde das Eingangsportal der Hochschule Merseburg ausgewählt. Es wird als 3D Objekt im ArchitectView dargestellt und es wird ein Willkommensgruß angezeigt, wenn man durch das Portal hindurch läuft.

```
poi.push({
    location: new AR.GeoLocation(51.343835, 11.976393),
    content: new AR.Model("assets/home_portal.wt3", {
        scale: {
            x: 1,
            y: 1,
            z: 1
        },
        rotate: {
            roll: 0.0,
            tilt: 0.0,
            heading: -24.8
        }
    }),
    indicator: true
});
```

Listing 4.13: main.js: POI als 3D Objekt

Dieses Objekt hat genauso wie die Labels eine `GeoLocation`, jedoch ist der `content` komplett verschieden. Anstatt dem `AR.Label` wird nun ein `AR.Model` für das 3D Objekt benötigt. Das erste Argument beim Instanziieren des `AR.Model` ist die Pfadangabe zum

3D Modell. Dieses 3D Modell wurde mit Blender[7] erstellt, als Autodesk Datei im `.fbx` Format exportiert und mit dem Wikitude 3D Encoder [15] in ein Wikitude kompatibles `.wt3` 3D Modell konvertiert.

Außerdem muss die Skalierung des 3D Modells angegeben werden. Da das Modell bereits maßstabsgetreu erstellt wurde, muss es nicht skaliert werden. Jedoch muss das Modell noch rotiert werden, da das virtuelle Objekt entlang der x-, y- und z-Achse ausgerichtet ist, das Eingangsportal aber nicht parallel zu einem Breitengrad ist. Das Objekt wird also um -24.8° um die z-Achse rotiert, damit die Ausrichtung mit der Realität übereinstimmt.

Visualisierung

Nachdem die Eigenschaften und Verhaltensweisen der virtuellen Objekte definiert wurden, müssen sie geladen und angezeigt werden. Dazu dient die Funktion `createGeoObjects()`.

```
1  function createGeoObjects(poi) {
2      var indicator = {};
3      indicator.image = new AR.ImageResource("assets/indi.png");
4      indicator.drawable = new AR.ImageDrawable(indicator.image,
5          0.1, {
6              verticalAnchor: AR.CONST.VERTICAL_ANCHOR.TOP
7          });
8      for (i = 0; i < poi.length; i++) {
9          poi[i].geoObject = new AR.GeoObject(poi[i].location, {
10             drawables: {
```

[7] Open Source 3D-Grafiksoftware.

```
                cam: [poi[i].content]
            }
        });
        if (poi[i].indicator) {
            poi[i].geoObject.drawables.indicator = indicator.drawable;
        }
    }
}
```

Listing 4.14: main.js: Objekte laden und anzeigen

Diese Funktion durchläuft das Array von `poi`-Objekten und instanziiert für jedes Objekt ein `AR.GeoObject` mit den während der Initialisierung definierten Eigenschaften. Wenn vorher so festgelegt, wird an dieser Stelle auch der Indikator zugewiesen.

Positionstrigger

Mit dieser Funktion wird das Verhalten erzeugt, dass eine Meldung angezeigt wird, wenn man durch das Eingangsportal läuft.

```
function createLocationTrigger(poi) {
    var arrived = false;
    AR.context.onLocationChanged = function(lat, lon, alt, acc) {
        if (poi[2].geoObject.locations[0].distanceToUser() < 5
                && !arrived) {
            alert("Willkommen an der Hochschule Merseburg");
            arrived = true;
        }
```

```
9       };
10  }
```

Listing 4.15: main.js: Positionstrigger setzen

Da sich das „Durchlaufen“ schwierig erkennen lässt, wurde die Erkennung etwas vereinfacht. Die Entfernung des Benutzers zum Objekt lässt sich viel einfacher bestimmen, deswegen wird diese zu Hilfe genommen. Wenn sich der Benutzer dem Objekt auf weniger als 5 m nähert, wird die Willkommensmeldung angezeigt.

Infotext anzeigen

Die Funktion `togglePoiDescriptionFullscreen()` sorgt dafür, den Infotext für den Polymerisationskessel POI anzuzeigen und wieder auszublenden.

```
1   window.togglePoiDescriptionFullscreen
2           = function togglePoiDescriptionFullscreen() {
3       $('#poiDescription').toggle();
4   }
```

Listing 4.16: main.js: Infotext anzeigen

Die ID `poiDescription` zeigt auf ein `div`-Element mit der Beschreibung im Body. Dieses `div` ist über CSS von vornherein ausgeblendet. Mit der jQuery Funktion `toggle()` wird die Sichtbarkeit des `div` umgeschalten. Also durch einen Klick auf den POI wird `togglePoiDescriptionFullscreen()` aufgerufen und das `div` sichtbar gemacht. Innerhalb dieses `div` befindet sich der Button „Schließen“, welcher auch die Funktion `togglePoiDescriptionFullscreen()` aufruft. Klickt man auf diesen Button, wird das `div` wieder ausgeblendet.

Die Funktion wird hierbei an das `window`-Objekt gehängt, um sie auch im globalen Scope verfügbar zu machen. Das ist notwendig, damit die Funktion über den Button im HTML wieder geschlossen werden kann.

4.5.4 Ausführung

Letztendlich werden die Funktionen der Reihe nach ausgeführt und jeweils das `poi`-Array übergeben.

```
1 initialisation(poi);
2 createGeoObjects(poi);
3 createLocationTrigger(poi);
4
5 $('#loadingMessage').remove();
```

Listing 4.17: main.js: Funktionen ausführen

Nachdem die Funktionen ausgeführt wurden, kann die „Laden“-Meldung entfernt werden. Die App hat jetzt alles fertig geladen, die Objekte werden im ArchitectView angezeigt und die Anwendung ist bereit für Benutzereingaben.

4.6 Testen der Anwendung

4.6.1 Performanz

Eine wichtige Anforderung an die Anwendung war die Performanz. Da das GPS Tracking gerade bei der Performanz Vorteile gegenüber dem Tracking mit Markern hat, sollte es hierbei keine Probleme geben. Beim Testen der Anwendung unter realen

Bedingungen konnte dies bestätigt werden. Die Anwendung lief auf allen getesteten Geräten, unter anderem dem OnePlus X und dem Nexus 5, flüssig und es konnten keine Verzögerungen festgestellt werden.

Einzig das Starten der Anwendung dauert auf allen Geräten einige Sekunden. Der Grund dafür ist das Wikitude SDK und der ArchitectView. Das SDK enthält große und umfangreiche Bibliotheken, welche lange zum Laden brauchen. Dazu kommt, dass große Teile des SDK in JavaScript geschrieben sind. Die Performanz von JavaScript hat sich zwar in den letzten Jahren mit verbesserten JavaScript Engines immer weiter gesteigert, jedoch gibt es noch viele Bereiche, in denen beispielsweise Java oder C++ Anwendungen deutlich schneller sind.

4.6.2 Genauigkeit der Sensoren

Ein weiterer wichtiger Aspekt ist die Genauigkeit der Sensoren. Hierbei konnte festgestellt werden, dass das GPS Modul eine sehr genaue Position auf der x- und y-Achse liefert. Beim Testen war keine Abweichung von der realen Position sichtbar. Es konnte aber eine leichte Verzögerung wahrgenommen werden. Die virtuelle Position hing der realen Position beim Bewegen immer um wenige Sekunden hinterher.

Jedoch bereitete die Messung der Höhe (Altitude) erhebliche Schwierigkeiten. Die Höhe konnte nur auf 15 m genau bestimmt werden und schwankte auch ohne reale Positionsänderung stark. Dadurch wurden die virtuellen Objekte im Himmel oder im Boden angezeigt. Dies ist besonders kritisch, da eine falsche Höhe viel stärker sichtbar ist, als eine falsche Entfernung. Deswegen wird der Altitude Wert des GPS Moduls nicht in der Anwendung verwendet, sondern die Höhen werden relativ zum Benutzer angegeben. Es wurde angenommen, dass sich das Gerät bei der Benutzung der App auf ca. 1,5 m Höhe befindet. Demzufolge werden alle Objekte auf der z-Achse bei -1,5

positioniert.

Zur Genauigkeit von Gyrosensor und digitalem Kompass lässt sich sagen, dass diese Sensoren genau und zuverlässig funktionieren. Es ist jedoch bei genauem Hinsehen ein leichtes Zittern aller Objekte in horizontaler Richtung sichtbar. Dies kann auf eine kleine Ungenauigkeit im digitalen Kompass zurückgeführt werden.

5 Zusammenfassung und Ausblick

Das Ziel der Arbeit war es, herauszufinden, wie weit das Tracking mit GPS fortgeschritten ist. Die in der Arbeit gewonnene Erkenntnis ist, dass das GPS Tracking bereits mit Einschränkungen in Augmented Reality Anwendungen verwendet werden kann. Die 2D Lokalisierung funktioniert sehr gut und sicher. Bei der 3D Lokalisierung müssen jedoch Abstriche gemacht werden. Das Problem bei GPS ist, dass sich die Höhe (Altitude) nur sehr ungenau bis gar nicht bestimmen lässt. Dadurch funktioniert die Positionierung auf der z-Achse nicht. Als Workaround werden alle Höhenangaben relativ zur Position des Betrachters gemacht. Eine Änderung der Position des Betrachters in z-Richtung kann so jedoch nicht berücksichtigt werden.

Angewendet werden kann diese Technik des Trackings z. B. bei Augmented Reality Spielen, die sich an Orten in der realen Welt abspielen. Dazu können 2D und 3D Objekte in die reale Welt projiziert werden, sodass die reale Welt zum Spielfeld wird. Diese Technik kann aber z. B. auch für eine neue Art von Filmen genutzt werden, bei denen man sich mitten im Geschehen befindet. Eine Möglichkeit der Realisierung kann dabei so aussehen, dass man einen bestimmten Schauplatz betritt und daraufhin in der Augmented Reality Anwendung die Akteure auftauchen und miteinander interagieren. Man selbst steht mitten im Geschehen und kann sich in der Szene frei bewegen.

Verbessern könnte man diese Methode mit zusätzlichen Sensoren, die die Umwelt

besser und genauer erfassen können. Google forscht bereits in der Richtung unter dem Namen „Project Tango“ [16]. Im Zuge von Project Tango werden Smartphones und Tablets mit Sensoren zur Tiefenabtastung ausgestattet. Damit können die Geräte alle Objekte in ihrem Umfeld erkennen. Mit solchen Sensoren könnte die Positionsbestimmung stark verbessert werden und eine Genauigkeit von wenigen Zentimetern liefern.

Abschließend lässt sich sagen, dass Augmented Reality noch relativ am Anfang steht. Die Technologien und Anwendungen werden sich in den nächsten Jahren weiterhin stark verbessern und Augmented Reality zu mehr Popularität verhelfen.

Abkürzungsverzeichnis

API Application Programming Interface. 15–17, 19, 22, 23, 28

CSS Cascading Style Sheets. 25, 26, 28, 36, 38, 45

FPS Frames per second. 23, 24

GPS Global Positioning System. 4, 5, 10, 14–16, 19, 20, 25, 33, 40, 46, 47, 49

HTML Hypertext Markup Language. 25, 26, 28, 35, 36, 38, 46

JS JavaScript. 25, 28, 36, 38

POI Point of Interest. 11, 20, 21, 38–42, 45, 53

QR Quick Response. 7, 8

SDK Software Development Kit. 4, 6, 13–17, 19, 21, 22, 24, 25, 32, 35, 36, 38, 40, 47, 52

TTFF Time to first fix. 25

XML Extensible Markup Language. 27, 31

Abbildungsverzeichnis

Listingverzeichnis

Literaturverzeichnis

[1] https://www.wikiwand.com/de/Erweiterte_Realit%C3%A4t
Zugriff: 01.02.2016

[2] https://www.quora.com/How-does-the-magnet-select-button-for-Android-Cardboard-work
Zugriff: 30.10.2015

[3] http://www.wikitude.com/products/eyewear/google-glass-augmented-reality-sdk/
Zugriff: 06.11.2015

[4] http://www.wikitude.com/products/eyewear/epson-augmented-reality-sdk/
Zugriff: 06.11.2015

[5] http://www.wikitude.com/products/eyewear/vuzix-augmented-reality-sdk/
Zugriff: 06.11.2015

[6] http://www.wikitude.com/developer/documentation/android
Zugriff: 06.11.2015

[7] http://thenextweb.com/apple/2014/06/02/apple-announces-swift-new-programming-language-ios/
Zugriff: 15.11.2015

[8] https://cordova.apache.org/
Zugriff: 15.11.2015

[9] http://www.wikitude.com/developer/documentation/android
Zugriff: 16.11.2015

[10] https://www.wikiwand.com/de/Bildwiederholfrequenz
Zugriff: 29.11.2015

[11] https://developer.android.com/about/dashboards/index.html
Zugriff: 11.12.2015

[12] https://jquery.com/
Zugriff: 17.01.2016

[13] https://www.wikiwand.com/de/Global_Positioning_System
Zugriff: 23.01.2016

[14] https://www.wikiwand.com/de/Tracking
Zugriff: 23.01.2016

[15] http://www.wikitude.com/external/doc/documentation/latest/htmlcss/encoder.html
Zugriff: 27.01.2016

[16] https://www.google.com/atap/project-tango/
Zugriff: 02.01.2016

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Merseburg, den 9. Februar 2016

Tobias Franke